Preprint
UCRL-JC-151771

STOMP: A Software Architecture for the Design and Simulation UAV-based Sensor Networks

E.D. Jones, R.S. Roberts, T.C.S. Hsia

This article was submitted to
IEEE International Conference on Robotics and Automation, Taipei, Taiwan, 05/12/03 – 05/17/03

October 28, 2002

Submitted to the IEEE International Conference on Robotics and Automation, May 12-17, 2003, Taipei, Taiwan

STOMP: A Software Architecture for the Design and Simulation UAV-based Sensor Networks

Erik D. Jones*[†]
edjones@ucdavis.edu

Randy S. Roberts[‡]
roberts38@llnl.gov

T. C. Steve Hsia[†]
hsia@ece.ucdavis.edu

[†]Dept. of Electrical & Computer Engineering, U.C. Davis, Davis, CA 95616 USA
[‡]Lawrence Livermore National Laboratory, Livermore CA 94550 USA

***Abstract*—This paper presents the Simulation, Tactical Operations and Mission Planning (STOMP) software architecture and framework for simulating, controlling and communicating with unmanned air vehicles (UAVs) servicing large distributed sensor networks. STOMP provides hardware-in-the-loop capability enabling real UAVs and sensors to feedback state information, route data and receive command and control requests while interacting with other real or virtual objects thereby enhancing support for simulation of dynamic and complex events.**

***Index Terms*— Autonomous vehicles, communication networks, multirobot cooperation, simulation software**

I. Introduction

The use of unmanned air vehicles (UAVs) in sensing applications is becoming increasingly important. As the cost of UAVs decrease, it becomes practicable to use several UAVs to service large distributed networks to maintain robust communications, and decrease latency. As this trend continues it is critical to have tools that allow designers to simulate, control and prototype networks that use multiple UAVs. Such tools would allow designers to rapidly test and develop complex behaviors, conduct trade-off studies, and test new algorithms and hardware. To address these design and simulation issues, the Simulation, Tactical Operations and Mission Planning (STOMP) architecture was developed. STOMP is an application and framework designed to study and operate sensor networks where UAVs are fundamental to the collection of data.

The framework for STOMP is designed around an object oriented architecture thereby allowing designers to adapt behaviors and algorithms of existing objects or assemble new objects rapidly and easily. As in [1], STOMP uses a visual editor to allow designers to assemble and configure each simulation. Designers may specify the state of every object in the system individually, in groups or globally and, using the graphical event configuration system, define events to trigger state changes at specified times in order to test complex behaviors and response to exceptions.

*Corresponding Author

A simulated environment, however, is not sufficient to completely test all aspects of UAV control, response and cooperation[2]. It is simply not possible to simulate all of the dynamics of a full UAV and distributed network system. However, through a communication subsystem, STOMP connects the virtual environment to real-world hardware systems. Command, control and state information is exchanged with real UAVs and sensors, creating a feedback mechanism to both test new algorithms within the virtual simulation as well as hardware and software implementation in real UAVs and sensors.

In section II, a high-level functional overview of STOMP is presented. Predefined UAV and sensor object behaviors and features are described. The communication, event and display controllers are also covered. In section III, the software framework, class structure, key algorithms and data structures are described. Future expansion, development and use of STOMP is described in section IV.

II. Functional Architecture

STOMP is designed to simulate UAVs, sensors and their interactions in a distributed sensor network under a variety of conditions. It is designed to easily implement control and cooperation architectures, and displays the reaction of the architecture to events that the designer can script into the simulation through a graphical interface. Through an internal communication controller, STOMP can feedback information from real UAVs and sensors using wireless Ethernet for data acquisition from sensors and a wireless serial interface for command, control and state information thereby providing hardware-in-the-loop simulations.

A functional block diagram of STOMP is illustrated in Figure 1. As shown in the figure, STOMP consists of the following main blocks: 1) Sensor and UAV objects (where sensor objects are depicted as circles, and UAV objects are depicted as hexagons); 2) a Communication Controller; 3) an Event Controller; and 4) a Display Controller. Details of the functions of these blocks are given in the proceeding subsections.

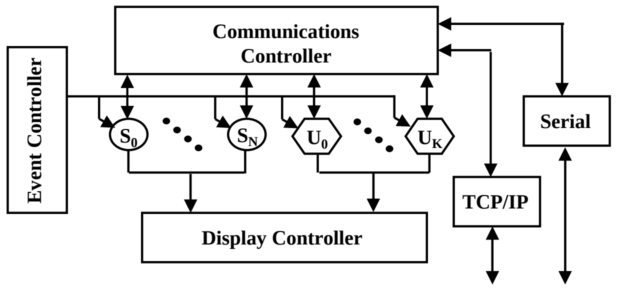

Fig. 1. Functional Block Diagram of STOMP

The UAV and sensor objects contain state information and algorithms relevant to the simulation and operation of UAVs and sensors in the network as given in Table I. Waypoints are the most fundamental building block of all STOMP simulation objects. The path planning controller and flight controllers use waypoints as means to control and calculate heading, position and plan routes. Waypoints contain only a single state variable, position. Position is stored in terms of decimal degrees of longitude and latitude. Depending on the configuration of each object, state

TABLE I

State	UAV	Sensor	Waypoint
Position	●	●	●
Altitude	●	●	–
Heading	●	●	–
Speed	●	●	–
Battery Life	●	●	–
Memory Usage	●	●	–
Status	●	●	–
Fuel	●	–	–
Vertical Velocity	●	–	–

updates may be obtained from the event controller (purely virtual), communication controller (purely real) or a combination (partially virtual). This feedback mechanism increases the richness of the dynamics that STOMP can simulate through the real and virtual interaction. Table II lists the static properties within the UAV and sensor objects. Static properties are set by the designer prior to running a simulation and define rates and characteristics used for communication and update of state information for pure or partially virtual UAVs and sensors.

TABLE II

Static Properties	UAV	Sensor
IP Address	●	●
Battery Capacity	●	●
Battery Drain Rate	●	●
Memory Capacity	●	●
Fuel Capacity	●	–
Fuel Drain Rate	●	–
Mechanical Characteristics	●	–

A. *UAV Objects*

UAV objects are equipped with a flight controller, communication controller and path planning controller as shown in Figure 2. If an external hardware device is not connected to that particular UAV, the flight controller simulates the dynamics of flight, providing position updates, changes in heading, ground speed, vertical velocity, fuel status and drain rate, and battery life and drain rate. If the flight controller is to be connected to a real UAV, the designer will set the appropriate flag when setting the initial states and parameters for the UAV using a graphical interface. This UAV is then connected via a wireless serial interface to a MP2000 autopilot engineered by Micropilot[3]. Using a communication wrapper, the MP2000 receives waypoint information from STOMP in order to control the direction of flight to service either virtual or real sensors. STOMP polls the MP2000 in order to update state information within the simulation asynchronously while the simulation is stepping through time.

The path planning controller implements the cooperation architecture described in [4] and path planning algorithm described in [5]. The path planning controller receives necessary information about the state of the system from other UAVs and sensors via the communication controller. This controller provides waypoint information to the flight controller in order to direct the UAV to the next sensor or waypoint.

B. *Sensor Objects*

Sensors contain their specific data acquisition equipment and a communication controller. Although STOMP has been designed to send and receive any kind of data, the display controller was designed to display and allow the user

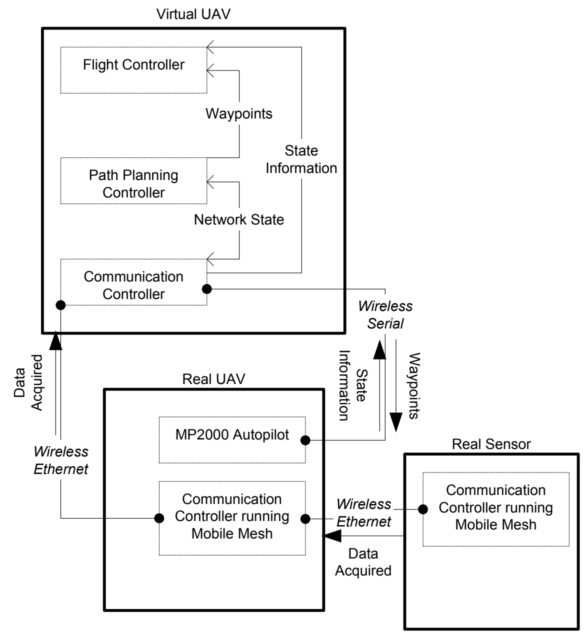

Fig. 2. Virtual UAV and Real UAV Communication

to access image data. The wireless Ethernet network in real sensors and UAVs is managed using Mobile Mesh[6] routing software and uses TCP sockets. STOMP operates at the application layer, while Mobile Mesh operates at layer 3 of the OSI model, thus STOMP remains independent of the specific communication and routing methods used in the real world environment.

Each sensor also contains a communication controller that implements a subset of the features included in STOMP. Since sensors only need to unload their data to UAVs and subsequently into STOMP, the data transfer protocol and ICMP functionality are the only features that are implemented in both simulated and real sensor elements.

C. Communication Controller

The communication controller is central to STOMP. It coordinates all communication between the main event controller, and real and virtual UAVs and sensors. Since this controller is the gateway to external hardware, it also routes command and control requests to the MP2000 autopilot via a wireless serial interface. The MP2000 is polled periodically and state changes are updated for the associated UAV object. The state information is then visualized within the simulator, providing the operator a graphical overview of the state of the system. As the simulation progresses, the communication controller will upload new waypoints to the MP2000 as determined by the path planning controller contained within the partially virtual UAV as shown in Figure 2.

As either virtual or real UAVs fly close to their target sensor node, the communication controller will attempt to contact the sensor using ICMP over a wireless Ethernet interface. If initial contact can be established, the UAV's communication controller will attempt to communicate with the sensor's communication controller in order to exchange data. The proprietary protocol uses a connected TCP socket to send and receive the appropriate data. The protocol can resume broken file transfers and provides full error checking in order to guarantee rapid and reliable transmission over possibly unreliable connections. If communication is lost, the communication controller attempts to contact the sensor again using ICMP and resume the file transfer process. If there was no new data available at the sensor or the data was successfully transferred, the communication controller notifies the path planning object and the UAV is sent to the next waypoint.

D. Event Controller

The event controller initializes the simulation, and provides several facilities for scripting simulation scenarios using a graphical interface. Scripting scenarios allows designers to study the reaction of the cooperation and path planning algorithms to various events. An event consists of a change in state (position, velocity, heading, etc.), disabling or enabling a sensor or UAV and the time at which the event occurs. STOMP tracks time in terms of steps which do not necessarily correlate with physical times. Events that occur externally are received by the communication controller and processed asynchronously to the event controller.

The event controller also provides facilities for post-simulation analysis of network communications. As the simulation progresses, all state information of every element in the simulation (sensors and UAVs) is recorded, along with line-of-sight data computed from the Digital Terrain Elevation Data (DTED) data. This state information can be exported to a file and is suitable for postprocessing by other analysis tools such as Matlab, OMNet++[7] or OPNET^{TM}. In this manner, the packet-level behavior of the network can be studied.

E. Display Controller

STOMP is divided into two different display modes, the designer view and the simulator view. When a new simulation is being created, the designer view is used to place objects within the coordinate space of the DTED space represented as a color coded topographical relief map. Labels may be added at various positions, enabling the identification of useful landmarks on the map. The user may either input the coordinates directly or use the mouse and graphical interface to place objects. While in the designer view, object properties and initial states may also be set.

When the simulation is started, the simulator view appears in front of the designer view. In this mode, display controller provides the designer with visual feedback of the state of the network as it progresses. It also displays data collected by operational UAVs from deployed sensors. The positions of the UAVs and sensors, along with the current UAV paths, are displayed on top of shaded terrain maps loaded from the DTED. When new data is received from a sensor, the sensor icon changes color, and the image is displayed by clicking on the indicator.

III. Software Architecture

STOMP was written in C++ using a highly object oriented design methodology. The modularity of the design allows for new objects to be designed and integrated into the framework rapidly. Objects are derived from base classes and provide updated functionality in order to support object-specific features. An overview of the class hierarchy is shown in Figure 3. Programmers may use inheritance to create more feature rich objects tailored to a particular application. All STOMP objects provide an initialization function and a Serialize() function for saving and restoring their state on disk.

A. *CSTOMPDoc - Scenario Class*

The CSTOMPDoc class contains all necessary information to start a simulation. UAVs, sensors, labels, waypoints, events and display parameters are stored in this class. CSTOMPDoc provides the mechanism for storing and retrieving state information for the scenario at time index 0. When a simulation is created, objects are copied out of this class into the simulation/event controller class for processing beyond time index 0.

When a simulation is ready for execution, CSTOMPDoc creates a new thread, opens a new view window and initializes the simulation and event controller. Control is then passed to the new view allowing the designer to begin testing the scenario.

B. *CSTOMPSim - Event and Simulation Class*

CSTOMPSim is the main event and simulation controller for the STOMP framework. After a scenario is designed, all of the object information defined by the user is

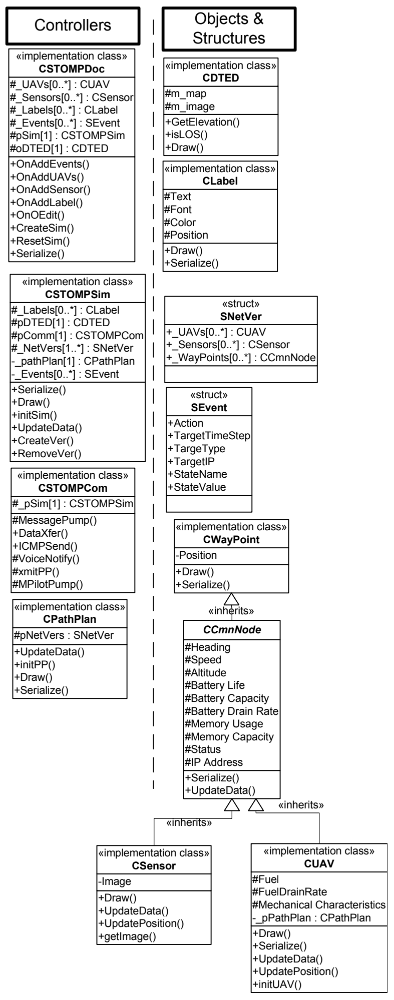

Fig. 3. UML Diagram of STOMP Framework

loaded into this class. CSTOMPSim then initializes all of the UAV and simulation objects, preparing them for synchronous state updates using the simulation clock which is controlled through the graphical interface.

CreateVer() is used to create the initial snapshot of the network for processing by the path planning controller in the leader UAV. As exceptions occur to the network, CreateVer() is used to create a new snapshot of the network for the leader UAV before transmitting its changes to the remaining UAVs that are controlled by CSTOMPSim. When the no more UAV objects are connected to a version, RemoveVer() is called to conserve memory and remove the stale version from the linked list.

CSTOMPSim creates a thread for the communication controller so that asynchronous updates to state information received from the MP2000 connected UAV or data received from external sensors can be processed independently of the simulation clock. This allows for the seamless integration of real and virtual devices without concerning the designer with timing issues. While a simulation can progress synchronously, a real UAV or sensor need not wait for STOMP's simulation clock pulse to continue with its next operation.

C. *CSTOMPCom - Communication Controller Class*

CSTOMPCom contains all of the logic to process communication requests received from the event controller, state updates or command and control to the MP2000 autopilot, and data transfer between UAVs and sensors, both virtual and real. The communication controller runs in an independent thread created by CSTOMPSim and can receive messages sent to its thread process, interrupts received from the MP2000 connected via a wireless serial interface or interrupts received via a TCP connected socket.

When an event occurs that requires the communication controller to take a particular action, such as to transmit new path planning information to all UAVs in the network, a message is sent to the process. Messages are processed in the MessagePump() function and action is taken to process the message and notify the event controller or target objects depending on the type of request received. CSTOMPSim, the UAV class (CUAV), and the sensor class (CSensor) provide mechanisms to lock internal data structures as needed, such as when processing asynchronous state updates received from external hardware. Data structure locking is the mechanism that prevents the asynchronous updates to interfere with the clocked events within CSTOMPSim.

Several functions within CSTOMPCom are called to take a particular action once a message has been properly decoded by MessagePump(). The DataXfer() function is called by a UAV to receive image data from real sensors after initial contact is verified using ICMPSend(). If new data has been received, CSTOMPCom will call VoiceNotify() to notify the operator using a synthesized voice that new data was received and identifies the sensor it was received from. If the leader UAV wishes to rebroadcast a new path plan, the UAV object will send a message which will trigger a call to xmitPP(). This function sends out new path plan and waypoint data to all UAVs in the network.

D. *CUAV, CSensor - Simulated Objects*

Simulated objects such as the UAV and sensors are contained in the classes CUAV and CSensor respectively. Each is derived from a common base class that contains behavior and information related to both objects and necessary for integration into STOMP. For example, every STOMP object displayed on the screen must supply a Draw() function and similarly, every object that is clocked within the simulator must provide an UpdateData() function. The interfaces for these functions is provided in the common base class. Simulated objects must also provide a Serialize() function for storing and retrieving its state information.

When the simulation clock triggers a call to UpdateData(), virtual UAVs and sensors update various state variables based upon static properties and rates set by the designer and dynamic models programmed into the function. UpdateData() calls UpdatePosition() which calculates the new position of the object based on its heading and speed. If it is a UAV object, the UpdatePosition() function checks to see if the UAV has arrived at a sensor and calls appropriate communication routines in CSTOMPCom to contact the sensor. Once CSTOMPCom has notified the UAV that its ready to proceed (after either a successful or unsuccessful attempt to communicate with the sensor), the UAV calculates a new heading to the next waypoint and proceeds.

Periodically, CSTOMPCom will call MPilotPump() to poll the MP2000 for new state information or to send new waypoint information if a MP2000 is associated with a UAV in the simulation. The virtual UAV that is connected to the MP2000 may trigger a call to MPilotPump() by sending the appropriate message to the CSTOMPCom thread if, for example, the network has been reorganized or the UAV needs to fly to another waypoint for some other reason.

E. *CPathPlan - Path Planning Class*

CSTOMPSim and every UAV object contains its own independent CPathPlan object. CPathPlan is the class that contains all of the cooperation and global path planning logic. During initialization, the path planning object is given a snap shot of the state of the network, including all UAV and sensor states. As needed, the path planning object will signal CSTOMPCom to transmit or retrieve information from real or virtual UAVs and sensors.

When exceptions in the network occur, such as the loss of a UAV or sensor, the communication object is first notified. It then creates a new version or snapshot of the network in CSTOMPSim. The leader[4] UAV is notified and given a pointer to the new state of the network. This new version is used by the leader to perform path planning and broadcast new commands and information to other UAVs. Like all STOMP simulation objects, main processing occurs in the clocked UpdateData() function.

IV. Conclusion and Future Work

In this paper we have presented an overview of the STOMP application and framework. As UAVs become less expensive to build and implement, these types of devices become easier to deploy in order to service large distributed networks. In order to increase robustness or to decrease latency, cooperation between UAVs becomes necessary. STOMP is a feature rich environment through which a designer may test new algorithms involving cooperation, communication, command or control of these networks. The graphical interface provides a mechanism to quickly setup purely virtual, partially virtual or purely real environments quickly and easily. When configured for purely real objects, STOMP essentially acts as a ground station, providing visual feedback to the operator of the state of the network and access to remote data.

Since it is impossible at this time to model all of the dynamics of a real system in a lab environment, the STOMP framework provides the ability to interact with external devices in real world deployments. The feedback from external UAVs and sensors has provided an extremely useful mechanism to more accurately test the behavior of large networks with very little hardware.

Thus far STOMP has been used as a command and control unit for the real UAVs by providing waypoint information to the MP2000 autopilot and processing the cooperative behaviors within the simulator (partially virtual). In the next phase of our research, the cooperative behaviors that have been modelled thus far in STOMP as described in [4] and [5] will be implemented in real UAVs. STOMP will be used to fully test the behavior of real UAVs and sensors by providing simulated communication and behavior making the UAV and sensors believe they are communicating with a much larger network.

Advancing the modularity and increasing the scripting features available to STOMP designers has potential benefits for a wide range of applications beyond simple UAV/sensor communication networks. In many areas of mobile robotics where large scale deployments or testing is necessary, STOMP can reduce the time and cost of designing these networks, testing complex behaviors and implementing those features through command and control or autonomy by providing a flexible framework that can interact with real devices and simulate larger environments.

Acknowledgements

This work was performed under the auspices of the U. S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48 and under IUT-B522313 from the Laboratory Directed Research and Development program. The authors would like to thank Dave McCallen, Director of the Center for Complex and Distributed Systems, for his support of the project. The authors would also like to acknowledge the support provided by Alan Smith, Chad Goerzen, Matt Myrick, and Noah Reddell.

References

[1] D. MacKenzie, R. Arkin, and J. Cameron, “Multiagent mission specification and execution,” *Autonomous Robots*, pp. 29–52, 1997.

[2] H. Hagras, V. Callaghan, and M. Colley, “Outdoor mobile robot learning and adaptation,” *IEEE Robotics and Automation Magazine*, vol. 8, pp. 53–69, September 2001.

[3] Micropilot Corporation, *MP2000 Autopilot*. [ONLINE] Available: http://www.micropilot.com.

[4] C. Kent and R. Roberts, “Cooperation architectures and adaptive path planning for unmanned aerial vehicles,” *Submitted to the 2003 IEEE International Conference on Robotics and Automation*, 2003.

[5] C. Cunningham and R. Roberts, “An adaptive path planning algorithm for cooperating unmanned air vehicles,” *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pp. 3981–3986, 2001.

[6] MITRE Research Group, *Providing Solutions For Mobile Adhoc Networking*. [ONLINE] Available: http://www.mitre.org/tech_transfer/mobilemesh.

[7] A. Varga, *OMNet++ Discrete Event Simulation Program*. [ONLINE] Available: http://www.hit.bme.hu/phd/vargaa/omnetpp.